



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels



Master's degree in Applications
and Technologies for
Unmanned Aircraft Systems

MASTER THESIS

TITLE: Using Drone Tello Edu for educational purposes

MASTER DEGREE: Master's degree in Applications and Technologies for Unmanned Aircraft Systems (Drones) (MED)

AUTHOR: Javier Alejandro Jimenez Cavadas

ACADEMIC ADVISORS: Miguel Valero Garcia and Pablo Royo Chic

DATE: October , 23rd 2019

Abstract

The scope of this project is the Tello EDU drone, both individually and as a swarm.

Its main use is for educational purposes. This project makes a study of how the state of art is. At the beginning, the visits to the Telopilots forum were very recurrent since it served me to see how other developers were working with this drone.

This project is responsible for gathering the information that allows anyone without prior knowledge to program a simple mission and also allows to know the development of more complex applications. As a final objective, a main application is proposed, consisting of the control of a swarm of drones with body movements.

Due to the disposition of the drones (initially I had only 1 drone, after a while I had the 4 drones), the tests and applications were first performed individually and at when I had the 4 drones, the swarm programming was carried out. That is why the document is also divided into these two variants.

To start programming the Tello EDU drone (Individual), it is based on a series of scripts already developed and functional, which are used as templates to develop the most complex applications.

After different tests with that script, I could create a script of a basic mission.

To start programming the swarm, an existing script is adapted (to control the drones it was done through a .txt file) so that it has a graphical interface of the style that I had used with the individual drone.

As in the case of the individual drone, this script provided me with the information necessary to create a basic mission.

For the main application, different experiments were carried out in relation to the processing of body movements. Two methods were used for this detection. The results and choice are contained in these experiments.

At the same time, other less complex applications were developed.

CONTENTS

INTRODUCTION	1
CHAPTER 1. STATE OF THE ART	3
CHAPTER 2. DRONE TELLO EDU	5
2.1. Technical Characteristics	6
2.2. Basic Environment before programming	8
2.3. Mission Pads.....	8
2.4. SDK 2.0	9
2.5. APP	9
CHAPTER 3. INDIVIDUAL DRONE.....	12
3.1. Example of basic mission.....	12
3.2. Commands that can be sent to the Tello EDU Drone	13
3.3. First Application: New commands created from the RC command	15
3.4. Main Application: Control of the Individual Drone with Body movements	20
3.4.1. First method: Open Pose.....	20
3.4.2. Second method: Colour Detection	23
3.5. Problems and failures	29
3.5.1. Lost messages	29
3.5.2. Non-continuous movements.....	29
CHAPTER 4. SWARM	30
4.1. Swarm definition.....	30
4.2. Swarm Environment	30
4.3. Swarm Setup	31
4.4. Example of basic mission.....	31
4.5. Secondary Applications.....	33
4.5.1. First Application: Swarm control in Real-Time	33
4.5.2. Second Application: Coordinates system	37
4.5.3. Third Application: Precise Take-off	41
4.6. Main Application: Control of the Swarm with Body movements.....	42
4.6.1. New Environment	42
4.6.2. New Setup	43

4.6.3. Preprogrammed Dances	44
4.6.4. Final Development.....	44
4.7. Discarded projects and causes	45
4.7.1. Letters.....	45
4.7.2. RC Commands	46
CONCLUSIONS.....	47
ACRONYMS	49
REFERENCES.....	50
ANNEX A: GITHUBS.....	51
ANNEX B: ACCURACY SAMPLE FIGURES.....	52
ANNEX C: ORG_COORD FUNCTION	53

INTRODUCTION

This project is focused on the Tello EDU drone. The Tello EDU drone, unlike its predecessor, the Tello drone, allows swarm programming in a very simple way, which is why the project will focus on both, its individual programming and its swarm programming.

The main use of this drone, as its name 'EDU' indicates, is for educational purposes. In this project, other drone platforms dedicated to education have been studied and how the Tello EDU drone is being used by other people or organizations. During this research, the GitHubs that are related to the Tello EDU drone and programmed in Python were also searched to see what things were to be developed and what things had been done. In addition, the visits to the Tellopilots forum were very recurrent since different developers exposed their progress and their doubts or problems, which has helped me in the future to avoid some mistakes.

As an initial objective for the project, it is intended to create some bases that allow anyone who has never programmed this drone, even without much programming knowledge, to perform some basic missions. These basic missions consist of taking off, taking a simple tour or a series of movements and landing. Other commands are also explained so anyone that follows the project can create more elaborate missions. As a final objective, so that the project has more travel, a complex application is developed that consists of controlling a swarm of drones through body movements. At the same time, other secondary applications that I considered could be useful for the main application were developed; but finally they were not included.

The drones used to perform the different experiments have been provided by the 'Universitat politecnica de Catalunya'. At the beginning they had only one drone, which is why I first began to familiarize myself with the programming of the individual drone, developing different applications but thinking about its possible adaptation for when I had the other drones. Having the 4 drones, the applications for swarm programming were focused. Due to this arrangement of drones, individual applications have been developed first and then for the swarm, which is why in the document there is this differentiation in the chapters.

To start programming the individual drone, and without ever programming a drone from scratch, I used a series of scripts already developed and focused on the control of one drone with body movements. These scripts are in a GitHub [1]. In this GitHub, you can find all the information (libraries that must be installed, description of the different scripts ...) necessary to execute the scripts.

The next step was to understand all the available scripts to continue developing the main application from that point. Understanding these scripts, then it had been very simple to create the basic mission.

For swarm programming, I also started from an already developed script; this GitHub [2] is referenced in the previous GitHub [1] at the end of the information,

for those who want to program a swarm. These Scripts are less developed and only allow the control of drones with a text file (.txt), which is why, with the knowledge previously acquired with the individual drone, I adapted it so that it has a graphical interface and the drones can be controlled from there.

Having to adapt the script, I had to understand it well beforehand, and this allows me to create the basic mission.

The development of the main application is also divided into an individual part and then already in the part with all drones. The individual part focuses on video processing in order to detect the position of the arms. First, the method that was already used in the used template scripts is used, which consists of the use of the 'Open Pose' library [3]. This library is very dependent on the graphics card, and like I am using a laptop with the 'Intel HD graphics' graphics card, it was observed that it took a long time to identify the position of the arms and different experiments were carried out to see if it could improve. After the results, I chose to use another method that required less processing; this new method consisted of the detection of colors through the 'OpenCV' library [4].

In the process of developing the main application, other applications that were intended to be used for the main application were being developed, however, in the end they were not implemented but are fully functional applications. Some of them appear in the document as secondary applications.

The remainder of the document is organized as follows. The first chapter presents the state of the art. Chapter 2 focuses on knowing in depth the Tello EDU drone, i.e technical characteristics, concepts before starting programming, accessories and others. Chapter 3 focuses on the programming of the individual drone, i.e basic mission, other commands, experiments for processing, problems and other applications. Chapter 4 focuses on swarm programming, i.e definition, setup, basic mission, final development of the main application and others. Finally a conclusion appears.

CHAPTER 1. STATE OF THE ART

Before starting the project, an investigation was carried out in order to learn about other platforms available in the market that allow us to take advantage of their educational potential. Different uses of the Tello EDU drone have also been investigated for the same purpose.

There are several drones that have similar characteristics and can be used for this purpose. The three most convenient for comparison have been selected below.

Parrot Mambo (Fig. 1.1): It is a drone developed by Parrot, whose characteristics are very similar to those of the Tello EDU drone. It allows programming in different languages, including Python and JavaScript, there is also a module in Simulink dedicated to this Drone. [5]



Fig. 1.1 Parrot Mambo

Robolink coDrone (Fig. 1.2): It is a drone developed by Robolink, which is especially focused on programming. Its characteristics are similar to those of the Tello EDU drone; however, it has an innovative frame that can attract the youngest people. It allows programming in Python and Arduino. [6]



Fig. 1.2 Robolink coDrone

Flybrix (Fig. 1.3): Unlike the drones mentioned above and the Tello EDU drone, this drone is buildable, without having to need extra tools, which adds a new step to learning. This drone can be programmed through Arduino. [7]



Fig. 1.3 Flybrix

Except for this last drone, the two previously mentioned have the option of block programming, just like in the Tello EDU.

The potential of Tello EDU for education is very large, and that is why there are several companies that are using it for these purposes. Among them we can highlight a few:

DrobotsCompany: It is a company that operates in several states of the United States. It is dedicated to courses for young people. Among its activities, it is worth mentioning the realization of missions with the Tello EDU drone, which makes young people have fun while they learn. [8]

Campuse: It is a company that operates in various parts of the world. They are specialized in giving courses related to new technologies. Its courses include the use of the Tello EDU drone to learn to program in Python and use OpenCV. [9]

Its easy programming has allowed developers to develop different applications:

DroneBlocks: It is an application that allows programming in a very simple way with blocks.

Tello FPV: It is an alternative application for drone control but it offers many more features than the application developed by RYZE. This application is not free.

Tellopilots is a forum where you can find many posts related to this drone. [10]

The GitHubs found in the research(together with a brief description of what is in each one) can be found in Annex A: GitHubs

CHAPTER 2. DRONE TELLO EDU

Tello EDU (Fig. 2.1) is a perfect programmable drone for educational use. In this section, the drone, its characteristics and other information necessary for the development of applications will be known in more depth.

Tello EDU is made by Shenzhen Ryze Technology and incorporate DJI flight control technology and Intel processors.

Before this version there was Tello, unlike this new model, it was white and did not allow swarm programming in a simple way.

This new version also has a new SDK which is more widespread than the previous one. [12]

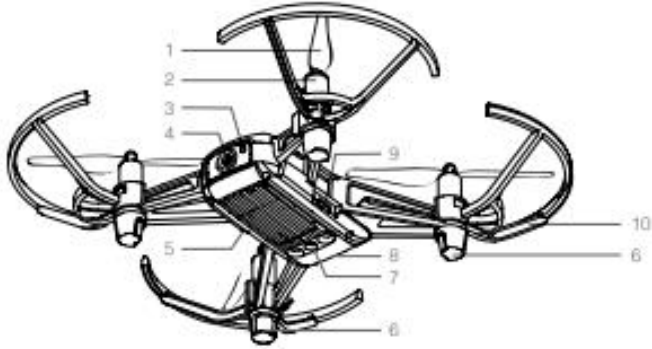


Fig. 2.1 Tello EDU

2.1. Technical Characteristics

The principal components of the drone are the following:

Table 2.1. Description of components of the Tello EDU drone

	1.Propellers
	2.Engines
	3.Drone Status Indicator
	4.Camera
	5.Power Button
	6.Antennas
	7.Vision Positioning System
	8.Battery
	9.Micro USB port
	10.Propeller protectors

Its dimensions and characteristics make it very manageable.

Table 2.2. Description of the aircraft

Weight	87 g
Dimensions	98x92.5x41 mm
Propeller	3 inches
Integrated Functions	Telemetric sensor
	Barometer
	LED
	Vision System
	Wi-Fi 2.4 GHz 802.11n
	Real-time streaming 720p
Port	USB battery charging port
Operating temperature range	from 0° to 40°
Operating frequency range	from 2.4 to 2.4835 GHz
Transmitter (EIRP)	20 dBm (FCC)
	19 dBm (CE)
	19 dBm (SRRC)

Being a drone for mainly indoor use, its characteristics that define its operations are quite limited.

Table 2.3. Details in operation

Maximum distance of flight	100 m
Maximum speed	8 m/s
Maximum flight time	13 min
Maximum flight height	30 m

It has a very easy battery to change; it can be charged directly with a USB cable with the battery inside the drone, or with a charging hub.

Table 2.4. Battery details

Removable	Yes
Capacity	1100 mAh
Voltage	3.8 V
Type	LiPo
Energy	4.18 Wh
Net Weight	25 ± 2 g
Temperature range when charging	from 5° to 45°
Maximum Load Power	10 W

The camera allows us to obtain video with a good quality (HD), after the image processing, it is possible to develop different applications.

Table 2.5. Camera details

Photo	5 MP (2592x1936)
Field of view	82.6°
Video	HD 720p 30 fps
Format	JPG (Photo)
	MP4 (Video)
Electronic stabilization	Yes

Vision Positioning System: Consisting of a camera and an infrared 3D module. This system is capable of working in a range of 0.3 m to 30 m high, but its optimal working conditions are 0.3 m to 6 m high.

Drone Status Indicator: It is a led that has the drone, and indicates in what state the drone is in each moment. (Turning on, without receiving messages...)

Field of view: It is the open observable area that the drone camera can see.

Electronic stabilization: It is an image enhancement technique using electronic processing.

2.2. Basic Environment before programming

To program the drone it is possible to use different operating systems (Windows, Linux, MAC ...), for this project, it has been programmed in Linux due to its ease to operate with the different libraries that were intended to be used.

It is possible to program in different languages (Java, C ++, Python, Scratch ...). Python has been used for this project.

To run a script it is necessary to connect the computer to the Tello EDU drone. To do this, we must connect to the Tello EDU Wi-Fi network. The name of the Wi-Fi network, by default, can be seen inside the drone when the battery is removed. It is possible to change the name of the network and also add a password to prevent any stranger from taking control of the drone.

2.3. Mission Pads

The Tello EDU drone comes with four tables. These tables are known as Mission Pads (Fig. 2.2). The Mission Pad is a 15x15 cm dimension table that serves as a pattern that can be identified by the Tello EDU drone. Each side contains a unique pattern composed of:

- Planets: different patterns that indicate the ID
- A rocket: follow the direction of the x-axis
- An ID: number between 1 and 8



Fig. 2.2 Mission Pads

The detection frequency of the cameras is 20 Hz if they are connected individually. If the two joints are connected, they have 10 Hz. The detection range for the Mission Pad by the Tello EDU drone is as follows (Fig. 2.3):

Detection Range

Height: 0.3 - 1.2 m

Range (height of 0.3 m): 0.4x0.4 m

Range (height of 1.2 m): 1x1 m

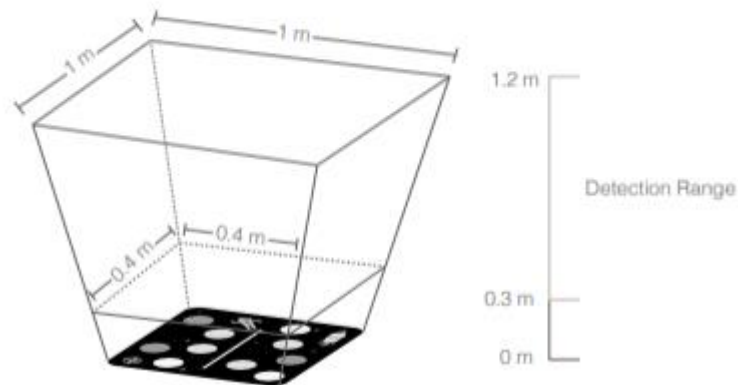


Fig. 2.3 Detection Range

2.4. SDK 2.0

The Tello EDU drone has a software development kit, which serves as the basis for the development of the different applications carried out in this document. On the internet, you can find a user guide [11] made by RYZE technologies in which it is explained how to make the connections, all the different commands that the drone accepts and how to reset it to return to the initial mode. These points are further explained throughout this document.

2.5. APP

The Tello EDU drone can be controlled through the Tello application of Shenzhen RYZE Tech Co. Ltd.

In the main interface (Fig. 2.4), we have both joysticks to control the drone and see what the drone's camera is capturing. In this same interface, we have data such as the battery, if it is connected to the Wi-Fi or Bluetooth, the height and the speed. We can also take photos and start recording the video.

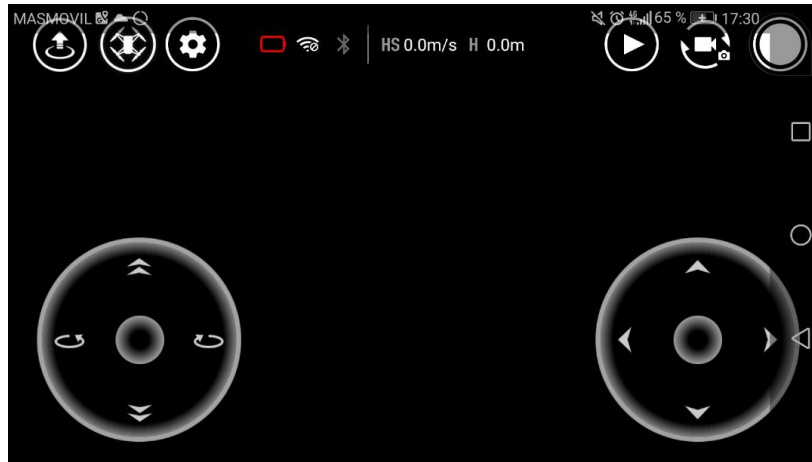


Fig. 2.4 Tello App main interface

Finally, we can access different functionalities such as takeoff and landing by sliding the screen (Fig. 2.5).



Fig. 2.5 Take-off functionality

And we can access the different flight modes (Fig. 2.6):

- Remote: Record a short video while the drone moves away and rises.
- 8D somersaults: Flip in 8 different directions.
- Circle: Record a short video while flying in a circle.
- 360°: Record a short video while rotating 360°.
- Throw in flight: You can throw the drone to a position in the air. It hovers there.
- Bounce mode: Fly automatically between 0.5 and 1.2 meters.



Fig. 2.6 Flight modes

CHAPTER 3. INDIVIDUAL DRONE

The objective of this section is to allow someone, who has no prior knowledge in the Tello EDU drone, to perform a simple mission by programming it through a Python script. After this, the different commands that can be send to the drone (all are included in the SDK) are developed a little more, and finally, different applications that have been made from these concepts are explained.

3.1. Example of basic mission

The first step is to connect the drone to the computer to be able to give orders through a Python script. Wi-Fi must be used to establish the connection between the drone and the computer. Then, commands are sent to take off, perform a mission (in this case a square) and land. (Fig. 3.1)

The commands that are used are explained with more depth in the section 3.2 Commands that can be sent to the Tello EDU Drone.

```
import socket

local_ip = ''
local_port = 9000

socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # socket for sending cmd
socket.bind(local_ip, local_port)

tello_ip = '192.168.10.1'
tello_port = '8889'
tello_addr = (tello_ip, tello_port)

socket.sendto(b'command', tello_addr) #start the commands

socket.sendto(b'takeoff', tello_addr)

socket.sendto(b'left 100', tello_addr) #perform a square
socket.sendto(b'backward 100', tello_addr)
socket.sendto(b'right 100', tello_addr)
socket.sendto(b'forward 100', tello_addr)

socket.sendto(b'battery?', tello_addr)
response, ip = socket.recvfrom(1024)
print response #receive the response from tello.

socket.sendto(b'land', tello_addr)

socket.close()
```

Fig. 3.1 Basic example Python script

First, we have to import the 'socket' library through which we will make the different connections.

After that, we define the local address:

- IP: 0.0.0.0
- Port: 9000

Having the local address, we create the socket and bind it.

The next step is to define the address of the Tello EDU drone:

- IP: 192.168.10.1
- Port: 8889

And we can use the socket that we created before, to send the different commands to this address. We will use the 'socket.sendto' python command to send the different commands. The arguments will be the tello command and the tello address.

The first command that has to be sent in order to be able to send other commands, is 'command', after this command we can use any of the ones that appear in the user guide. [11]

Then in the script, the take-off is performed, and a square is made in the air, following that order:

- left
- backward
- right
- forward

The 100 followed by the direction orders, is the distance in centimeters that we want the drone to move.

The script also shows how the response is received from the Tello EDU drone. In this example, we want to know the percentage of battery that the drone has at that time. For this, we use the 'battery?' command. The answer will be a value between 0 and 100, and it is obtained from the socket, with the python command 'recvfrom'.

Finally, the drone lands and the socket is closed.

3.2. Commands that can be sent to the Tello EDU Drone

This section will explain the different type of commands that can be sent to the Tello EDU Drone, and within these types, some commands are explained (all commands appear in the user guide [11] mentioned above)

Commands can be organized as follows:

- Control commands: They are the basic commands to control the drone.
- Set commands: They are used to set new sub-parameter values.

- Read commands: They are used to know the current value of the sub-parameters.

Control Commands:

Table 3.1. Control Commands

Command	Description
command	Enter SDK mode
streamon	Enable video stream
emergency	Stop motors immediately
takeoff	Auto Takeoff
up x	Ascend to 'x' cm. $20 < x < 500$
cw x	Rotate 'x' degrees clockwise. $0 < x < 360$
flip x	Flip in 'x' direction. 'l','r','f','b' l: Left r: Right f: Forward b: Backward
go x y z speed	Fly to 'x' 'y' 'z' at 'speed'. $-500 < x,y,z(\text{cm}) < 500$. $10 < \text{speed}(\text{cm/s}) < 100$ 'x,y,z' are the direction
stop	Hovers in the air. Note: works at any time
go x y z speed mid	Fly to 'x' 'y' 'z' of the 'mid' at 'speed'. $-500 < x,y,z < 500$. $10 < \text{speed} < 100$ 'x,y,z' are the direction 'mid' is the number of the mission pad

Set Commands:

Table 3.2. Set Commands

Command	Description
speed x	Set speed to 'x' cm/s. $10 < x < 100$
rc a b c d	Set remote controller control via four channels These commands are like the orders we can send using the joysticks of a controller. Therefore, we can vary four fields; these fields range from -100 to 100, and indicate speed: <ul style="list-style-type: none"> • a = Speed on the x axis • b = Speed on the y axis

	<ul style="list-style-type: none"> • c = Speed on the z axis • d = Roll speed <p>In this command the distance to be traveled is not indicated, that is why the 'stop' command is necessary to stop the movement.</p>
wifi ssid pass	<p>Set Wi-Fi password.</p> <p>ssid = updated Wi-Fi name. pass = updated Wi-Fi password</p>
mon	Enable mission pad detection
ap ssid pass	Set the Tello to station mode, and connect to a new access point, with the access point 'ssid' and 'pass'

Read Commands:

Table 3.3. Read Commands

Command	Description
speed?	Obtain current speed (cm/s). $10 < \text{speed} < 100$
battery?	Obtain current battery percentage. $0 < \text{battery} < 100$
time?	Obtain current flight time
wifi?	Obtain Wi-Fi snr
sdk?	Obtain the Tello SDK version
sn?	Obtain the Tello serial number

Safety feature:

The Tello EDU drone will land in case it is without receiving an order 15 seconds. For Real time applications it is recommended to create a new function that send the 'command' command (which after sending it for the first time to accept the other commands, then has no use) every 5 or 10 seconds in order to avoid interruptions.

3.3. First Application: New commands created from the RC command

Having seen the different commands and after different tests with them, it can be seen that there is some limitation as to the realization of some displacement or that there are commands whose use is very restrictive. That is why as a first application, the development of different displacements is carried out using the 'rc a b c d' command seen above.

Thanks to this command, other commands have been created such as a circle, arc or diagonal movement, which were not included in the SDK 2.0 or their implementation was limited.

The first function that is created is that of the circle, and serves as the basis for creating the other figures.

This is achieved by dividing a circle at different points all with the same separation distance.

I calculate the position of the different points of the circumference through which the drone should move (Fig. 3.2). This is achieved by knowing the angle (the angle is determined according to the number of divisions we want the circumference to have) and the radius. The speed value will be used to know the time interval between commands.

To calculate the angle, having the divisions, apply:

$$\emptyset = \frac{360}{divisions} \quad (3.1)$$

And having the angle we can calculate the position (x, y):

$$P_x = Radio * \cos \emptyset \quad (3.2)$$

$$P_y = Radio * \sin \emptyset \quad (3.3)$$

And the time between the different orders is calculated by:

$$t = \frac{2*radio}{speed} * \sin \emptyset \quad (3.4)$$

Thus a series of points of the circumference is obtained which has as its center the position (0,0). Then each point obtained from the circumference corresponds to the speed on the x and y axes that must be applied to the RC command so that from the point at which it is at each moment, it reaches the next. The arguments of this function are speed, divisions and radius. The function returns the points and the time interval.

This is applied to achieve a complete circumference, but it serves as a basis to achieve other figures:

- The arc displacement is achieved by choosing the start and end points of the arc. (Fig. 3.3)
- The diagonal displacement can be obtained by selecting the angle (it has to be multiple of the angle obtained through the divisions) and the distance. Time is calculated by dividing the distance by speed. (Fig. 3.4)

```
def calc_pts_circ(self, velocidad, divisiones, radio):  
    points = []  
  
    angulo_grad = 360 / divisiones  
    angulo_rad = (angulo_grad * math.pi) / 180  
    p0x = radio  
    p0y = 0  
    p0z = 0  
    points.append((p0x, p0y, p0z))  
  
    tiempo = (2.0 * float(radio) / float(velocidad)) * math.sin(float(angulo_rad) / 2.0)  
  
    for item in range(divisiones):  
        punto = item + 1  
        px = radio * math.cos(punto * angulo_rad)  
        py = radio * math.sin(punto * angulo_rad)  
        points.append((round(px), round(py), 0))  
  
    return points, tiempo
```

Fig. 3.2 calc_pts_circ python function

```
def arco(self, velocidad, divisiones, radio, p_inicial, p_final):  
    points, tiempo = self.calc_pts_circ(velocidad, divisiones, radio)  
    nuevos_points = []  
  
    if p_inicial > p_final:  
        p_final = p_final + divisiones  
  
    for item in range(p_inicial, p_final + 1):  
        posicion = item  
  
        if posicion >= divisiones:  
            posicion = posicion - divisiones  
  
        nuevos_points.append(points[posicion])  
  
    return nuevos_points, tiempo
```

Fig. 3.3 arco python function

```
def diagonal(self, velocidad, divisiones, radio, angulo_direcc, distancia):
    points, tiempo = self.calc_pts_circ(velocidad, divisiones, radio)
    point = []
    angulo_div = self.div_ang(divisiones)
    posicion = angulo_direcc/angulo_div
    point = points[posicion]
    tiempo_nuevo = float(distancia) / float(velocidad)
    return point, tiempo_nuevo
```

Fig. 3.4 diagonal python function

With these points obtained and the time interval between the commands, a function is created to send the RC commands to execute the forms. (Fig. 3.5)

The differentiation for the length of the points (greater or less than 3) is because for the diagonal displacement, only one point (x, y, z) is obtained and must be processed differently.

```
def enviar_comando_rc(self, points, tiempo):

    if len(points) > 3:
        for item in points:
            x = item[0]
            y = item[1]
            z = item[2]
            self.tello.comando_rc(x, y, z, 0)
            time.sleep(tiempo)
    elif len(points) <= 3:
        x = points[0]
        y = points[1]
        z = points[2]
        self.tello.comando_rc(x, y, z, 0)
        time.sleep(tiempo)
    self.tello.stop()
```

Fig. 3.5 enviar_comando_rc python function

All displacements are made in the xy plane by default, which is why a function to change coordinates is created (Fig. 3.6)

```
def cambio_coord(self, points, plano):  
    points_nuevos = []  
    if len(points) > 3:  
        for item in points:  
            x = item[0]  
            y = item[1]  
            z = item[2]  
  
            if plano is 'xz':  
  
                x_nueva = x  
                y_nueva = 0  
                z_nueva = y  
  
                points_nuevos.append((int(x_nueva), int(y_nueva), int(z_nueva)))  
            elif plano is 'yz':  
  
                x_nueva = 0  
                y_nueva = x  
                z_nueva = y  
  
                points_nuevos.append((int(x_nueva), int(y_nueva), int(z_nueva)))
```

Fig. 3.6 cambio_coord python function

The following example (Fig. 3.7) corresponds to a mission in which the drone executes a circle in the xy plane, then an arc in the xz plane and finally a diagonal towards the 210° direction in the xy plane.

```
points, tiempo = self.calc_pts_circ(30, 24, 30)  
self.enviar_comando_rc(points, tiempo)  
points, tiempo = arco(30, 24, 30, 4, 8)  
points = self.cambio_coord(points, 'xz')  
self.enviar_comando_rc(points, tiempo)  
points, tiempo = diagonal(30, 24, 30, 210, 40)  
self.enviar_comando_rc(points, tiempo)
```

Fig. 3.7 Mission example

3.4. Main Application: Control of the Individual Drone with Body movements

The main application to be developed in this project is the control of a swarm of drones with body movements.

To get the final result of our application, I first have to focus individually. What I want to achieve in this section is that the individual drone be able to process the video in real time to know the position of the arms at each moment.

First, the Open Pose method is used, whose script is already quite developed. This Script can be obtained from a GitHub [1]. That GitHub has served as a template for the development of the other applications. All library installation processes and other issues are well explained in that GitHub.

Seeing that the results are very poor and erratic, it is changed to a simpler method that involves the detection of colors.

3.4.1. First method: Open Pose

For this method, it is the Open Pose library [3], and we can use it through a script that recognizes the different parts of the body and lists them as shown in the image below (Fig. 3.8). We can define different positions to which we must then add only the command we want to perform.

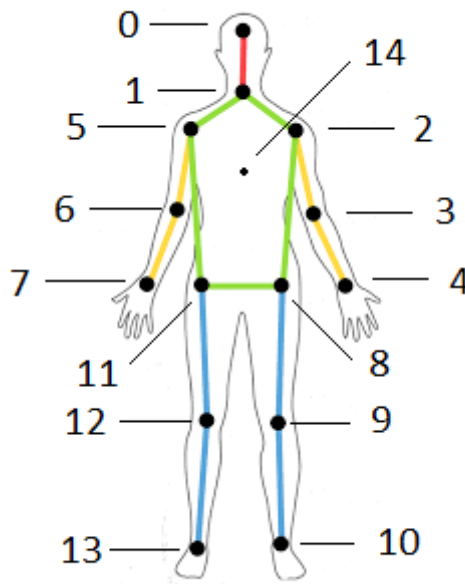


Fig. 3.8 Pose Model

At first, this was the method chosen for the detection of body positions, but due to the results obtained, in terms of time and accuracy, this method was discarded. The tests and results are set out below.

The problem comes because the library that processes the data is very heavy and dependent on the graphics card of the computer.

The objective of these experiments is to check if the image processing to obtain the position of the arms with the Open Pose library is achieved in a correct and fluid way with the computer that I am using to develop the project and the possible benefits of using a computer with a more powerful graphic card.

As the processing time is always going to be greater than 0, we cannot consider that a purely continuous movement will be achieved, however we will consider it as acceptable response time as long as it does not exceed 1 second.

This experiment has been developed with two different graphics cards:

- Intel HD Graphics
- GeForce GTX 1060 6GB

And the experiment had been carry out in the Laboratory 21 where the bottom is gray and is not very homogeneous because there are cabinets.

For this experiment, the three keypoint body models that allow the use of Open Pose have been used. These keypoint body models represent the different parts of the body to identify:

- MPI: 15 Points
- COCO: 18 Points
- 25-BODY: 25 Points

These models can be downloaded from the Open Pose github [3]. The three models need to have assigned the total points of the keypoint body model and their joining lines, necessary for the representation of the results. (Fig. 3.9)

```

MODE = "MPI"

if MODE is "COCO":
    self.nPoints = 18
    self.POSE_PAIRS = [ [1,0], [1,2], [1,5], [2,3], [3,4], [5,6], [6,7], [1,8],
                        [8,9], [9,10], [1,11], [11,12], [12,13], [0,14], [0,15], [14,16], [15,17]]

elif MODE is "MPI" :
    self.nPoints = 15
    self.POSE_PAIRS = [[0,1], [1,2], [2,3], [3,4], [1,5], [5,6], [6,7],
                      [1,14], [14,8], [8,9], [9,10], [14,11], [11,12], [12,13]]

elif MODE is "BODY_25" :
    self.nPoints = 25
    self.POSE_PAIRS = [[1,8], [1,2], [1,5], [2,3], [3,4], [5,6], [6,7], [8,9], [9,10],
                      [10,11], [8,12], [12,13], [13,14], [1,0], [0,15], [15,17], [0,16],
                      [16,18], [2,17], [5,18], [14,19], [19,20], [14,21], [11,22], [22,23], [11,24]]

```

Fig. 3.9 Models and joining lines

In addition to the keypoint body models, another variable can be modified in order to obtain different results. This variable are the InWidth and InHeight values, that can be modified in the 'tello_pose.py' script [1]. The higher the InWidth and InHeight values, the greater the accuracy, but the processing time will also be longer.

For the experiments, because the drone's battery lasts a short time in flight, the drone has been placed on an elevated position to capture the entire body without having to take off. The position to be evaluated by the drone is with the arms extended. Being the most basic position, it has been enough to get the results we were looking for. The last column determines which models could be considered for this project; those with a response time of less than one second and an acceptable or better accuracy will be valid.

Table 3.4 Results with the 'GeForce GTX 1060 6GB' graphics card

Model	InWidth InHeight	Time(s)	Accuracy	Valid
MPI	100	0.184	Null	No
MPI	150	0.382	Bad	No
MPI	200	0.611	Acceptable	Yes
MPI	250	0.996	Very good	Yes
MPI	300	1.375	Very good	No
MPI	350	1.846	Very good	No
COCO	100	0.256	Bad	No
COCO	150	0.536	Acceptable	Yes
COCO	200	0.882	Acceptable	Yes
COCO	250	1.380	Very good	No
COCO	300	1.906	Very good	No
BODY-25	100	0.438	Bad	No
BODY-25	150	0.832	Acceptable	Yes
BODY-25	200	1.423	Very good	No

Table 3.5. . Results with the ‘Intel HD Graphics’ graphics card

Model	InWidth InHeight	Time(s)	Accuracy	Valid
MPI	200	2.575	Null	No
MPI	250	3.174	Bad	No
MPI	300	4.739	Acceptable	No
MPI	350	6.938	Very good	No
COCO	200	3.439	Bad	No
COCO	250	5.154	Acceptable	No
COCO	300	7.923	Very good	No
BODY-25	150	3.631	Bad	No
BODY-25	200	5.640	Acceptable	No
BODY-25	250	8.148	Very good	No

The first table corresponds to the experiment performed on the laboratory computer that has the ‘GeForce GTX 1060 6GB’ graphics card. The second table corresponds to the experiment carried out with my laptop (Intel HD Graphics).

For the computer with the graphics card ‘GeForce GTX 1060 6GB’ there are some valid models, however, this computer was used daily by a partner, and for the development of the project it was decided to change the detection method.

Four sample figures that represent the different levels of accuracy can be found in the Annex B: Accuracy sample figures.

3.4.2. Second method: Colour Detection

After seeing that the results obtained by the Open Pose method depended in large part on the graphics card of the computer (In this project I use a laptop) and that they were not good enough (in terms of processing time and accuracy) the method is replaced by color detection. With this new method, we avoid the problem of continuity since its processing is very fast.

The method of color detection, as the name implies, is to recognize a certain color in an image or video. For this, the Python OpenCV library [4] is used.

The method used in this project to replace body movements is to identify the position (coordinate) of 3 different colors (one will go in the right hand, the other in the left and the third will be fixed in the chest) to be able to represent body movements. The positions of the hands can be up, middle or down with which we

can obtain up to 9 different positions (ex: right hand up, left hand middle) (Fig. 3.10)

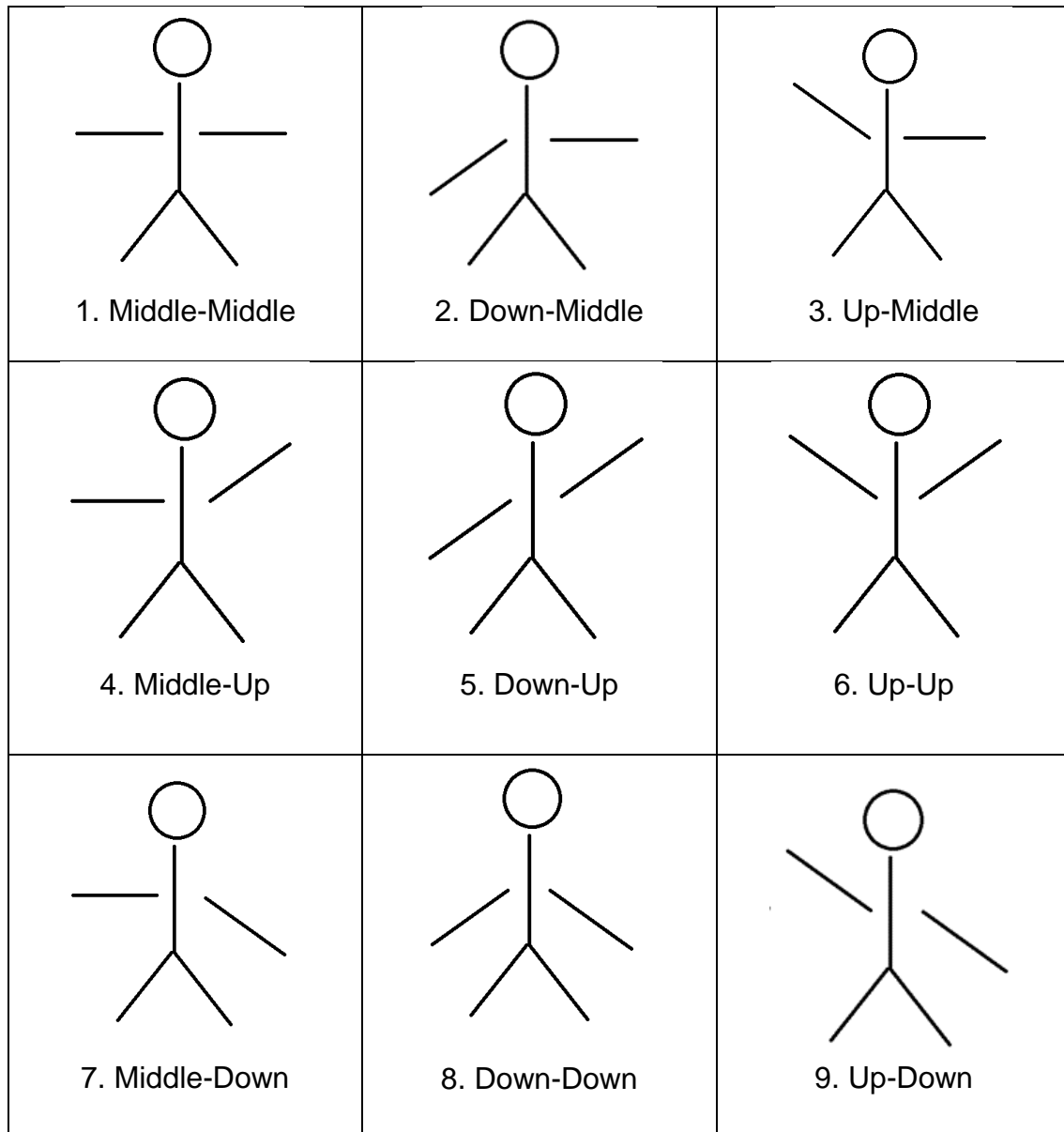


Fig. 3.10 Arms positions

The first thing to do is define what colors I want to identify. In this project the colors green, red and blue are used because they are easy to identify without the appearance of other threats (similar colors that slip into the video and provide incorrect data), although due to the changes of light these threats can occur.

To detect the colors, filters are applied. The images received after the 'streamon' command, are received in BGR (Blue-Red-Green) format and we need to convert them to HSV (Hue-Saturation-Value) format using the command line 'cv2.COLOR_BGR2HSV'. The range of HSV values is what we need to know to identify each color.

To obtain these thresholds, tests must be carried out in order to adjust them correctly.

It is necessary to create a new function that will not have use for the control of the drone but that serves to be able to perform the tests with greater ease. This function consists of a trackbar with the different adjustable values to obtain the desired color filter (Fig. 3.11).

The tests are done focusing with a drone to the wall of the laboratory, in which the cards of the different colors are placed or on other occasions the cards were held by me (Fig. 3.12), and go adjusting by trial and error until the cardboard of the color that we want to delimit, is the only thing that is shown in the tab of results (Fig. 3.13). There are other forms of representation, but it is the most easy to see the result.

```
def trackbar(self):

    cap = self.frame
    cv2.namedWindow("Trackbars")
    cv2.createTrackbar("L - H", "Trackbars", 0, 179, self.nothing)
    cv2.createTrackbar("L - S", "Trackbars", 0, 255, self.nothing)
    cv2.createTrackbar("L - V", "Trackbars", 0, 255, self.nothing)
    cv2.createTrackbar("U - H", "Trackbars", 179, 179, self.nothing)
    cv2.createTrackbar("U - S", "Trackbars", 255, 255, self.nothing)
    cv2.createTrackbar("U - V", "Trackbars", 255, 255, self.nothing)
    while True:
        frame = self.frame
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        l_h = cv2.getTrackbarPos("L - H", "Trackbars")
        l_s = cv2.getTrackbarPos("L - S", "Trackbars")
        l_v = cv2.getTrackbarPos("L - V", "Trackbars")
        u_h = cv2.getTrackbarPos("U - H", "Trackbars")
        u_s = cv2.getTrackbarPos("U - S", "Trackbars")
        u_v = cv2.getTrackbarPos("U - V", "Trackbars")
        lower_blue = np.array([l_h, l_s, l_v])
        upper_blue = np.array([u_h, u_s, u_v])
        mask = cv2.inRange(hsv, lower_blue, upper_blue)
        result = cv2.bitwise_and(frame, frame, mask=mask)
        cv2.imshow("frame", frame)
        cv2.imshow("mask", mask)
        cv2.imshow("result", result)
        key = cv2.waitKey(1)
        if key == 27:
            break

    cv2.destroyAllWindows()
```

Fig. 3.11 Trackbar function



Fig. 3.12 Detection test

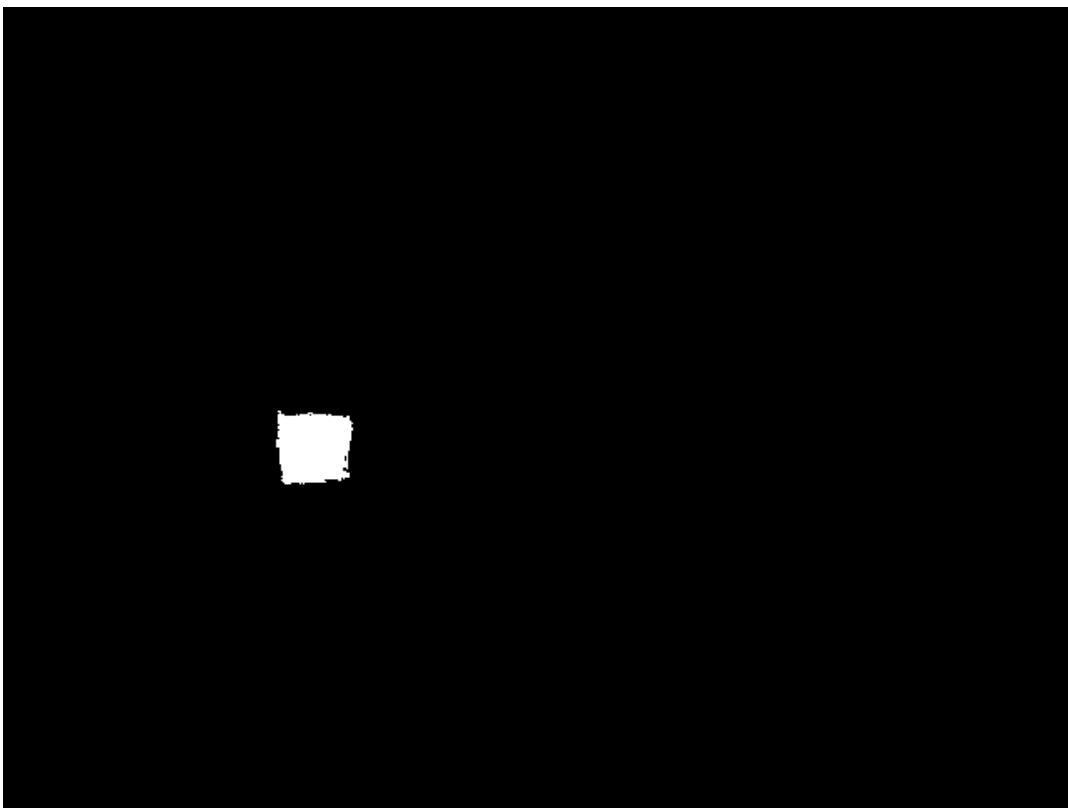


Fig. 3.13 Detection result

Having the six values identified, we go on to define the color in a function to use it later. (Fig. 3.14) For this, we define the color range and it find the area where it appears in the image (the condition that the area be greater than 20 is so that no threats are sneaked) and finally we get the position on the 'y' axis (it is the only value that will interest us to identify your position)

```
def detect_colour_rojo(self, image):

    x=0
    y=0
    w=0
    h=0

    rojo_bajos = np.array([125, 120, 132], dtype = "uint8")
    rojo_altos = np.array([179, 255, 255], dtype = "uint8")

    red = cv2.inRange(image, rojo_bajos, rojo_altos)

    contours, hierarchy=cv2.findContours(red,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

    for pic, contour in enumerate(contours):
        area = cv2.contourArea(contour)

        if(area>20):
            x,y,w,h = cv2.boundingRect(contour)
            print 'rojo'
            print 'x: %d, y: %d, w: %d, h: %d' % (x,y,w,h)
            self.red = y

    return x,y,w,h
```

Fig. 3.14 detect_colour_rojo function

To obtain the different positions you will have to take into account the position of the colors that we have in the hands, in reference to the one in the chest. Calculating it on the 'y' axis if:

- Lateral colour \geq Center colour + 40 cm -> UP
- Lateral colour \leq Center colour - 40 cm -> DOWN
- (Lateral colour < Center colour + 40 cm) and (Lateral colour > Center colour - 40 cm) -> MIDDLE

This is defined with a function called 'Posicion_Brazos', shown below for the left arm (Fig. 3.15). In case of not detecting the color in the image, the value assigned to the position it is (0,0), that is why you have to label the arm as 'None' when it is in that position, so later it is not represented.


```

def Posicion_Brazos(self,Circulo_iz,Circulo_der,Circulo_cen):

    Brazo_iz = 'None'
    Brazo_der = 'None'
    altura_var = 40

    if (Circulo_iz >= (Circulo_cen + altura_var)) and (Circulo_cen is not 0) :

        Brazo_iz = 'Abajo'

    elif (Circulo_iz <= (Circulo_cen - altura_var)) and (Circulo_iz is not 0):

        Brazo_iz = 'Arriba'

    elif (Circulo_iz is 0) or (Circulo_cen is 0):

        Brazo_iz = 'None'

    else:

        Brazo_iz = 'Centro'

```

Fig. 3.15 Posicion_brazos function

With the different positions defined, we will assign to each position a different movement. When a position is identified, until a new one is identified, the movement is only performed once.

A position is defined through the position of the right arm and the left arm. When you enter that position, the command you want to execute is sent and the value of that position is assigned to a variable, so that the same action is not repeated.

Given the impossibility of moving from a position with the arms down to a position with the arms up without going through another position, a support position is created.

This support position is 'Middle-Middle' to which no movement will be assigned and will serve to repeat movements.

The assigned positions and movements are as follows:

- Position 1: support position(no command)
- Position 2: left 100
- Position 3: backward 100
- Position 4: forward 100
- Position 5: flip r
- Position 6: flip f
- Position 7: right 100
- Position 8: land
- Position 9: flip l

3.5. Problems and failures

After performing the different tests and trials, different problems had been observed (independent of the Python script) which lead to incorrect drone behavior.

3.5.1. Lost messages

Although messages are always sent, there are times when the drone does not execute the action. In the tests it was found that at the beginning of the operations the loss of messages is much smaller compared to the loss of messages that can occur at the final of the operations. When you touch the drone you can notice the heat it gives off. Therefore it follows that drone overheating causes message losses. This problem leads to errors in the synchronization of swarm movements.

3.5.2. Non-continuous movements

After performing the same test several times, having 2 reference points (mission pads) and moving the same distance, it was found that it is not always at the same point after the same displacement. This problem leads to synchronization between drones, not being so visually beautiful, or that after a certain step, the error of where it should be is greater.

CHAPTER 4. SWARM

4.1. Swarm definition

A swarm of drones consists of a large number of robotic systems, which are capable of performing collective behavior that emerges from the interactions between unmanned aerial vehicles and their interactions with the environment.

The basic idea of the UAV swarm is to make decisions by themselves through the information they share with each other.

Inspired by swarms of bees or flocks of birds, they would be able to perform various tasks both in national security areas and for commercial areas.

Being able to control all drones as if they were one, allows to occupy more territory. This is one of the biggest advantages over other existing technologies for search and rescue missions. [13]

4.2. Swarm Environment

In this Project, the elements used to configure the swarm environment are as follows (Fig. 4.1):

- 4 x Tello EDU Drones
- Laptop
- Router

Tello EDU Drones will be our main protagonists, responsible for carrying out the operations indicated.

The laptop is where the data is processed and the orders are given to the drones through a Python script.

The router serves as a junction point between the computer and the drones, all these elements are connected to the router. (In section 4.3 it is explained in more detail)

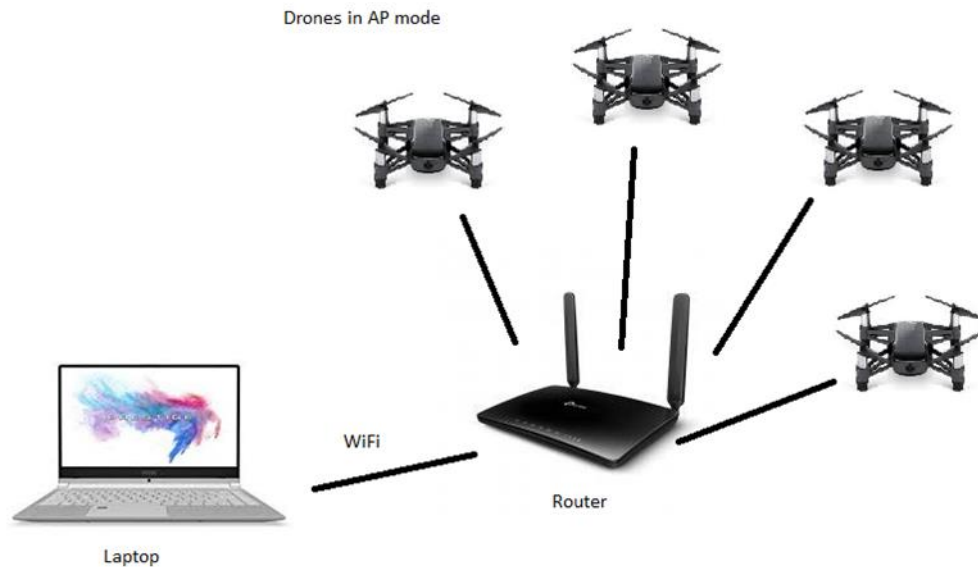


Fig. 4.1 Swarm environment

4.3. Swarm Setup

To configure the environment, what we have to do is connect the drones to the router. For this, we must set the drones in station mode. This is achieved by sending the 'ap ssid pass' command:

- ssid: Wifi name (router)
- pass: Wifi password (router)

This process is done individually for each drone. To send this command, first connect the computer to the drone that we want to set in station mode, the next step is to send the command 'command' and finally send the 'ap ssid pass' command (in the multi-formation github [2] there is a script ready to run and perform this task). The router assigns an IP (type 192.168.10.x) to the drone automatically.

When the drone is in station mode, we cannot connect to its network from the computer, to return to the initial mode we must reset the drone by pressing the power button for 5 seconds with the drone turned on.

4.4. Example of basic mission

This example (Fig. 4.2), with a swarm, is similar to the one already explained in the individual drone section.

```

self.local_ip = ''
self.local_port = 8889
self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # socket for sending cmd
self.socket.bind((self.local_ip, self.local_port))

Ip_Drone1 = '192.168.10.150'
Ip_Drone2 = '192.168.10.151'
Ip_Drone3 = '192.168.10.152'
Ip_Drone4 = '192.168.10.153'

socket.sendto(b'takeoff', (Ip_Drone1, 8889))
socket.sendto(b'takeoff', (Ip_Drone2, 8889))
socket.sendto(b'takeoff', (Ip_Drone3, 8889))
socket.sendto(b'takeoff', (Ip_Drone4, 8889))

socket.sendto(b'up 100', (Ip_Drone1, 8889))
time.sleep(1)
socket.sendto(b'up 100', (Ip_Drone1, 8889))
time.sleep(1)
socket.sendto(b'up 100', (Ip_Drone1, 8889))
time.sleep(1)
socket.sendto(b'up 100', (Ip_Drone1, 8889))

time.sleep(1)
socket.sendto(b'down 100', (Ip_Drone1, 8889))
time.sleep(1)
socket.sendto(b'down 100', (Ip_Drone1, 8889))
time.sleep(1)
socket.sendto(b'down 100', (Ip_Drone1, 8889))
time.sleep(1)
socket.sendto(b'down 100', (Ip_Drone1, 8889))

socket.sendto(b'land', (Ip_Drone1, 8889))
socket.sendto(b'land', (Ip_Drone2, 8889))
socket.sendto(b'land', (Ip_Drone3, 8889))
socket.sendto(b'land', (Ip_Drone4, 8889))

socket.close()

```

Fig. 4.2 Basic example mission

As in the first example, the first necessary thing to do is to define the local address with its port in order to create the socket through which to send the commands, in this case through the router.

Then, we will define the address of the drones, whose IP has been assigned directly to us by the router when setting the drones in AP mode.

As in the first example, the Python command 'socket.sendto' is used to send the different commands.

In that example:

- all four drones at once take off
- with a difference of one second, the drones will rise 100 centimeters
- with a difference of one second, the drones will descend 100 centimeters
- all four drones at once land

4.5. Secondary Applications

Two scripts are used to control the swarm:

- In the first one, the class that sends the different commands to the different drones is created; the script 'Tello_manager' is downloaded from the multi-formation github [2] and little modified since it fulfilled what was needed for this project.
- The second one is obtained using as a template the one used for the individual drone (eliminating unnecessary functions), and taking as a reference the other Script that appears in the multi-formation github [2], 'multi_tello_test.py', which worked by creating a text file with the different commands that we wanted to send to the different drones (making impossible to do changes once the script has been executed). The Script made by me allows real-time control.

I consider important to include the function created to send the commands to the different drones, because it will be used in the different secondary applications and in the main application.

There are two versions of this function, the initial one was to send the same command to the drones, in the second version 'save_action2' (Fig. 4.3) it is possible to send different commands through a list.

The arguments of this function are the list of drones and the list of commands to be executed by the different drones. An execution pool is created.

```
def save_action2(self,id_list,list_action):  
  
    it = 0  
  
    for tello_id in id_list:  
        tello_id = tello_id - 1  
        tmp_sn = self.id_sn_dict[tello_id]  
        reflec_ip = self.sn_ip_dict[tmp_sn]  
        fid = self.ip_fid_dict[reflec_ip]  
        # se crea la lista de los comandos que se tienen que enviar  
        self.execution_pools[fid].put(list_action[it])  
        it = it + 1
```

Fig. 4.3 save_action2 function

4.5.1. First Application: Swarm control in Real-Time

The first application that is developed consists of a graphical interface (Fig 4.4) to control the swarm of drones in real time. This application allows you to send one by one the commands we want to carry out or save a series of commands and then play them at once.

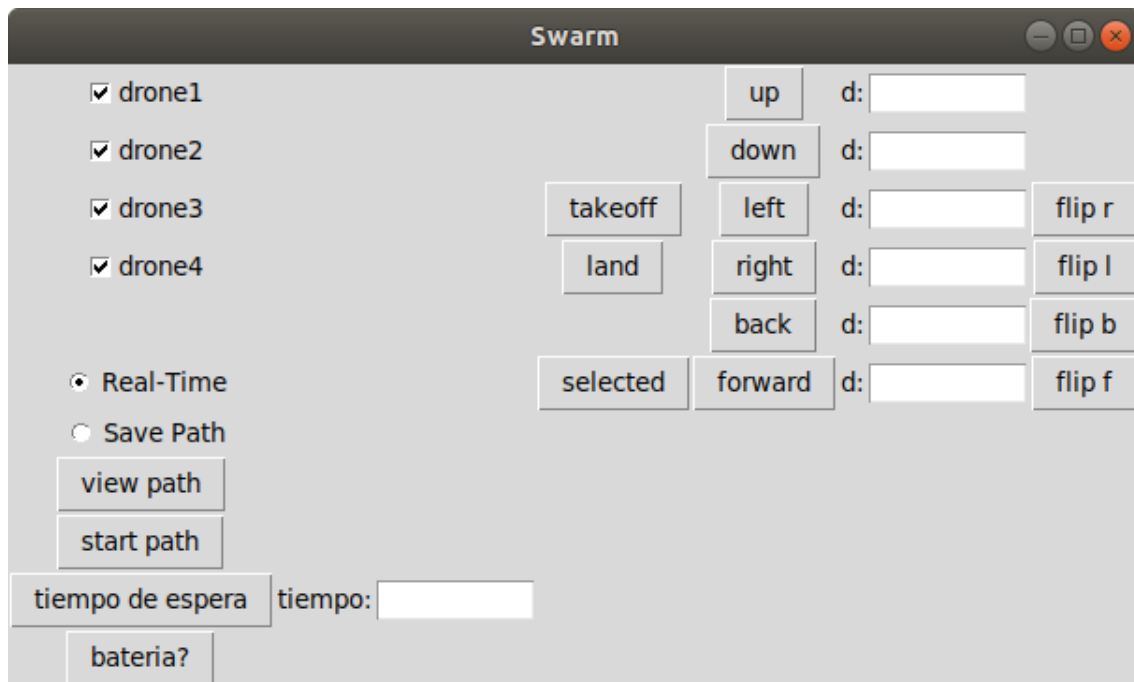


Fig. 4.4 Graphical interface of the first application

It is possible to select which drones we want to control at every moment. This is achieved creating an empty list, and when selecting the drones that we want to control at that precise moment, the list is filled with the number that corresponds to the selected drone (it can be 1, 2, 3 or 4). (Fig. 4.5)

```
def tello_selec(self):
    lista_selec = []

    if self.chk_state_d1.get() is 0:
        lista_selec.append(1)
    if self.chk_state_d2.get() is 0:
        lista_selec.append(2)
    if self.chk_state_d3.get() is 0:
        lista_selec.append(3)
    if self.chk_state_d4.get() is 0:
        lista_selec.append(4)

    return lista_selec
```

Fig. 4.5 tello_selec function

In a similar way it is possible to define the functions to obtain the values of the command that we want to carry out. The following image represents the move down command (Fig. 4.6)

```
def move_down2(self):

    distancia = self.down.get()
    comando = 'down %d' % int(distancia)
    lista = self.tello_selec()

    if self.selected.get() is 1:
        self.save_action(lista, comando)

    elif self.selected.get() is 2:

        self.lista_comandos.append(comando)
        self.lista_drones_selec.append(lista)
```

Fig. 4.6 move_down2 function

The '.get ()' command that appears in the tello_selec and move_down2 functions is responsible for taking the value that is set in the graphical interface

And in order to work with these functions through the graphical interface, it will be necessary to create different buttons such as those defined below:

- The self.selected button, initially set to 1 (Fig. 4.7), which would be Real Time, is responsible for differentiating Real Time and Save Path. (Fig. 4.8)

```
self.selected = tki.IntVar()
self.selected.set(1)
```

Fig. 4.7 self.selected initiation and set

```
self.rad1 = tki.Radiobutton(panel, text='Real-Time', value=1, variable=self.selected)
self.rad2 = tki.Radiobutton(panel, text='Save Path', value=2, variable=self.selected)

self.rad1.grid(column=0, row=5)
self.rad2.grid(column=0, row=6)
```

Fig. 4.8 Real-Time and Save Path buttons

- The button to select the selected drones will be defined as follows. In that case for the Drone 1. (Fig. 4.9)

```

self.chk_state_d1 = tki.IntVar()

self.chk_state_d1.set(0)

self.check_d1 = tki.Checkbutton(panel, text='drone1',var=self.chk_state_d1,onvalue=0, offvalue=1)
self.check_d1.grid(row = 0, column = 0)

```

Fig. 4.9 Drone 1 selection button

- The button for the command and its variables. In that case for the command move down. (Fig. 4.10)

```

self.btn_down = tki.Button(panel, text="down", relief="raised", command=self.move_down2)
self.btn_down.grid(row = 1, column = 4)

down_dist = tki.Label(panel, text='d:')
down_dist.grid(row = 1, column = 5)

self.down = Entry(panel,width = 10)
self.down.grid(row = 1, column = 6)

```

Fig. 4.10 Command down button

Depending on the actions in real time, you can:

- Send a command individually to be executed by the selected drones
- Save a series of commands with the delay times between commands and specifying to which drone is applied and then execute this series of commands as if it were a predefined plan (it is also possible before executing it, see a log of what is going to do)

In case it is in real time, the command is sent directly to the selected drones. In the case that it is a preprogrammed route, an empty command list is created in which the different commands to be executed will be saved, a list of drones selected for each command to be executed will also be created. Finally, a list of waiting times is created, which will be the times between the execution of each command. In case of not selecting any waiting time between commands, these will be executed after the last one is finished. For this case, the `start_path` functions are created to start sending the commands and `shown_path` to show the predefined route before executing it. (Fig. 4.11)


```

def shown_path(self):
    pos = 0
    for item in self.lista_comandos:
        pos = pos + 1
        print 'Accion %d: los drones %s --> %s' %(pos,tuple(self.lista_drones_selec[pos-1]),item)

def start_path(self):
    pos = 0
    pos_espera = 0

    for item in self.lista_comandos:
        lista = self.lista_drones_selec[pos]
        comando = item
        print lista
        print comando
        self.save_action(lista,comando)
        pos = pos + 1

    if pos is self.lista_posicion_esperas[pos_espera]:
        time.sleep(int(self.lista_esperas[pos_espera]))

        if (pos_espera+1) < len(self.lista_posicion_esperas):
            pos_espera = pos_espera + 1
            print pos_espera
            print len(self.lista_posicion_esperas)

```

Fig. 4.11 shown_path and start_path functions

4.5.2. Second Application: Coordinates system

In this project, a coordinate system has been created to move the drones to known points in space. Initially, it was necessary to take into account the crossings between drones so that there were no collisions; later an algorithm (drones that obstruct the path, first move back 100 cm, and after making the horizontal and vertical displacements, they move forward 100 cm) has been developed to be able to execute the orders without having to think about the crossings.

First, a function that is responsible for combining the displacements from one point to another with the displacements in centimeters is defined, 'move_to_Point' (Fig. 4.12). The distance from one point to another corresponds to 100 cm.

```
def move_to_Point(self, Point_ini, Point):  
  
    POx = Point_ini[0]  
    POz = Point_ini[1]  
  
    Pfx = Point[0]  
    Pfz = Point[1]  
  
    horiz = Pfx - POx  
    vert = Pfz - POz  
  
    if horiz > 0:  
  
        posiciones = horiz * 100  
        comando_hor = 'left %d' % posiciones  
  
    elif horiz < 0:  
  
        posiciones = horiz * -100  
        comando_hor = 'right %d' % posiciones  
  
    elif horiz == 0:  
  
        comando_hor = 'None'
```

Fig. 4.12 move_to_Point function

Next, it is necessary to create a function 'org_coord' to handle these displacements, from one point to another, for the different drones that we select. The full function with the algorithm to solve de crosses (because of its length, the images of that part of the function are not included) is in the Annex C: org_coord function.

We will need a list of endpoints, which will be the positions we want our drones to reach. The initial points of each drone will be:

- Drone 1: (1,1)
- Drone 2: (2,1)
- Drone 3: (3,1)
- Drone 4: (4,1)

First, the different points (initial and final) are individualized since they are in lists. (Fig. 4.13)

```
if len(Lista_drones) is 4:
    P01 = self.Lista_pnts[0]
    P02 = self.Lista_pnts[1]
    P03 = self.Lista_pnts[2]
    P04 = self.Lista_pnts[3]

    Pf1 = Lista_pnts_fin[0]
    Pf2 = Lista_pnts_fin[1]
    Pf3 = Lista_pnts_fin[2]
    Pf4 = Lista_pnts_fin[3]
elif len(Lista_drones) is 3:
    P01 = self.Lista_pnts[0]
    P02 = self.Lista_pnts[1]
    P03 = self.Lista_pnts[2]

    Pf1 = Lista_pnts_fin[0]
    Pf2 = Lista_pnts_fin[1]
    Pf3 = Lista_pnts_fin[2]
```

Fig. 4.13 obtaining points

Then, horizontal and vertical commands are assigned with the previously created function 'move_to_point'. After making a move to another point, this last point will be saved as a new starting point in order to assign another movement from that same point. In case it is not going to move on that axis, the command is saved as 'None' (Fig. 4.14)

```
if 1 in Lista_drones:
    comando_horiz1, comando_vert1 = self.move_to_Point(P01,Pf1)
    self.Lista_pnts[0] = Pf1
else:
    comando_horiz1 = 'None'
    comando_vert1 = 'None'

if 2 in Lista_drones:
    comando_horiz2, comando_vert2 = self.move_to_Point(P02,Pf2)
    self.Lista_pnts[1] = Pf2
else:
    comando_horiz2 = 'None'
    comando_vert2 = 'None'

if 3 in Lista_drones:
    comando_horiz3, comando_vert3 = self.move_to_Point(P03,Pf3)
    self.Lista_pnts[2] = Pf3
else:
    comando_horiz3 = 'None'
    comando_vert3 = 'None'

if 4 in Lista_drones:
    comando_horiz4, comando_vert4 = self.move_to_Point(P04,Pf4)
    self.Lista_pnts[3] = Pf4
else:
    comando_horiz4 = 'None'
    comando_vert4 = 'None'
```

Fig. 4.14 Commands assignment

The next step is to create four different lists:

- List of drones that will move in the vertical plane (Fig. 4.15)
- List of commands for vertical movement (Fig. 4.15)
- List of drones that will move in the horizontal plane
- List of commands for horizontal movement

```
if comando_vert1 is not 'None':  
    drones_vert.append(1)  
    comandos_vert.append(comando_vert1)  
  
if comando_vert2 is not 'None':  
    drones_vert.append(2)  
    comandos_vert.append(comando_vert2)  
  
if comando_vert3 is not 'None':  
    drones_vert.append(3)  
    comandos_vert.append(comando_vert3)  
  
if comando_vert4 is not 'None':  
    drones_vert.append(4)  
    comandos_vert.append(comando_vert4)
```

Fig. 4.15 Creation of vertical lists

Finally, the commands are sent (Fig. 4.16). Vertical versus horizontal movement is prioritized because all drones start from the same horizontal axis.

```
# se realizan los movimientos verticales y horizontales  
  
self.save_action2(drones_vert,comandos_vert)  
self.save_action2(drones_horiz,comandos_horiz)  
time.sleep(1)
```

Fig. 4.16 commands sent

4.5.3. Third Application: Precise Take-off

Mission Pads are one of the elements that comes with the Tello EDU drone. The main objective of these Mission Pads is to make more precise movements with marks on the ground.

One good application is to better align drones after takeoff. To do that the following function is used (Fig. 4.17):

```
def good_position(self):  
  
    Lista_drones = [1,2,3,4]  
    comando1 = 'go 50 50 100 30 m1'  
    comando2 = 'go 50 50 100 30 m2'  
    comando3 = 'go 50 50 100 30 m3'  
    comando4 = 'go 50 50 100 30 m4'  
    lista_comandos =[comando1,comando2,comando3,comando4]  
  
    self.save_action2(Lista_drones,lista_comandos)
```

Fig. 4.17 good_position function

The command makes the drone go to a position based on the Mission Pad.

4.6. Main Application: Control of the Swarm with Body movements

In this section, we want to continue with the progress made in section 3.4. Main Application: Control of the Individual Drone with Body movements, using the environment described for the swarm. However, a problem was found related to the reception of the video that necessitated a change in the environment.

The problem is that it is not possible to receive the video transmission, from any drone, when the drones are connected to the router (When they are in AP mode).

The impossibility of receiving video when the drones are in AP mode does not allow to fulfill the initial objective of this project, that is why a compromise solution is searched. That solution consists of the combination of an individually connected drone to the computer (being able to obtain the video from this drone) and other 3 drones connected to the Router.

4.6.1. New Environment

The elements used to configure the swarm environment are as follows (Fig. 4.18):

- 3 x Tello EDU Drones(connected to the Router)
- 1 x Tello EDU Drone(connected to the Laptop)
- Laptop
- Router
- Wi-Fi Adapter

In that case, the 3 Tello EDU Drones connected to the Router will be our main protagonists, responsible for carrying out the operations indicated. In addition, the individual drone will be like our camera and it is responsible for capturing the video.

The Wi-Fi Adapter will be necessary to have two connections on the laptop (the individual drone and the router)

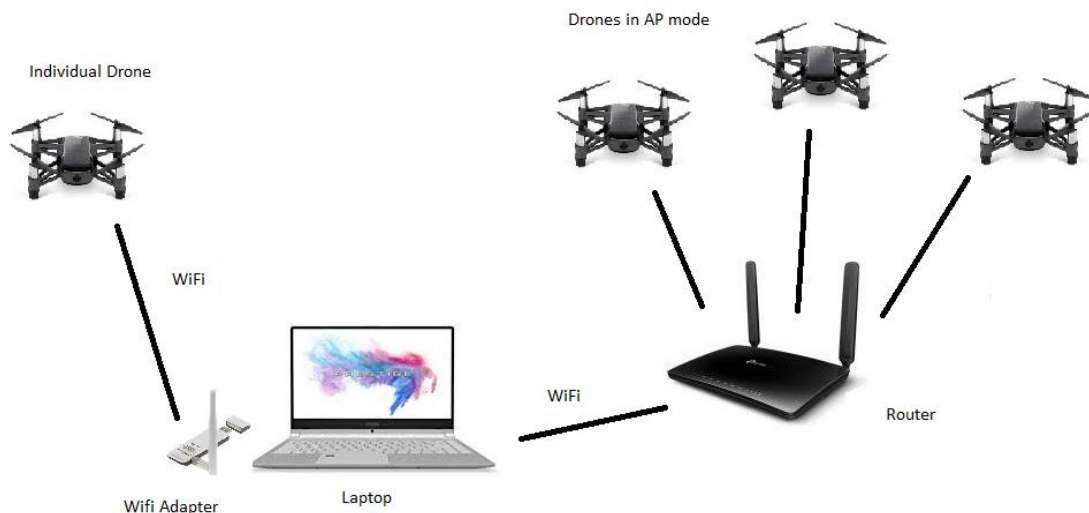


Fig. 4.18 New Swarm environment

4.6.2. New Setup

In this case, because the IP of the individual drone is 192.168.10.1 and that of the router and the drones connected to it are of the range 192.168.10.x, it will be necessary to take a few steps before to perform a correct routing, since initially only one of the two networks worked even if the computer was connected to both.

The steps are the following:

- Differentiate the two connections with its port (Fig. 4.19)

```
self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # socket for sending cmd
self.socket.setsockopt(socket.SOL_SOCKET, 2, 'wlp3s0')
self.socket_video = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # socket for receiving video stream
self.socket_video.setsockopt(socket.SOL_SOCKET, 2, 'wlp3s0')
```

Fig. 4.19 Python lines to differentiate the connections

- Linux: Execute the following commands in the terminal for proper routing.
 - `sudo ip route add 192.168.10.1/32 dev wlx18d6c70cbd4c`

wlp3s0 and wlx18d6c70cbd4c are the names associated to the port of connection of the laptop and the one of the wifi adapter respectively.

4.6.3. Preprogrammed Dances

For that section we will need to have a series of movements (let's call them dances) already programmed for the different body positions. To create a dance you can use the procedures explained above. They are created as a succession of different commands.

You can create any dance that enters the imagination. The following are created in the script (Fig. 4.20):

- Wave: Drones emulate a wave (rising and falling in coordination)
- Rotation: Drones rotate forming a square

```
def Bailes_Movimiento(self, posicion):  
  
    drones = [1,2,3]  
  
    if (posicion == 2) and (self.repetido is not 2) :  
        self.save_action(drones, 'takeoff')  
        self.repetido = 2  
  
    if (posicion == 7) and (self.repetido is not 7) :  
        self.save_action(drones, 'land')  
        self.repetido = 7  
  
    if (posicion == 3) and (self.repetido is not 3) :  
  
        self.Ola_de3()  
        self.repetido = 3  
  
    if (posicion == 4) and (self.repetido is not 4) :  
  
        self.Cuadrado_alante()  
        self.repetido = 4
```

Fig. 4.20 Bailes_Movimientos function

Positions 2 and 7 are linked to take off and land respectively

4.6.4. Final Development

The main script will be similar to the one in the Swarm section but updated for only 3 drones and adding the function that starts the individual drone (Fig. 4.21), which is achieved by executing the first script.

With this, we have a drone that records and captures the movements and the other 3 drones that will react according to the positions that the individual drone has captured.

```
def Iniciar_Mom(self):  
  
    drone_madre = tello.Tello('', 9000)  
    self.vplayerS = TelloUI(drone_madre, "./img/")  
  
    # start the Tkinter mainloop  
    self.vplayerS.root.mainloop()
```

Fig. 4.21 Iniciar_Mom function

4.7. Discarded projects and causes

There are some projects or applications that have to been removed due to it incompatibilities with a good development of the final objective.

4.7.1. Letters

The objective was to form letters with the drones and after this, words or phrases.

For this, the drones were initially taken to an initial formation and from this initial position, the necessary commands were sent to form the letter and then return to the initial formation (Fig. 4.22). (This was done so as not to have to create a new procedure for each letter after another letter)

The first problem comes from forming the letters with only 4 static points, to solve this it was thought to use movements to determine a line of the letter.

The second problem and which was ruled out is that the formation of some letter was achieved by the superposition of 3 drones in the vertical plane on the same coordinate 'y'; which produced many stability problems with drones.

```
def J(self):  
  
    drone2 = [2]  
  
    comando_drone2_1 = 'up 120'  
    comando_drone2_2 = 'left 120'  
    comando_drone2_3 = 'go 0 0 300 50 m3'  
  
    self.save_action(drone2,comando_drone2_1)  
    self.save_action(drone2,comando_drone2_2)  
    self.save_action(drone2,comando_drone2_3)  
  
def volver_J(self):  
  
    drone2 = [2]  
  
    comando_drone2_1 = 'right 120'  
    comando_drone2_2 = 'down 120'  
    comando_drone2_3 = 'go 0 0 200 50 m2'  
  
    self.save_action(drone2,comando_drone2_1)  
    self.save_action(drone2,comando_drone2_2)  
    self.save_action(drone2,comando_drone2_3)
```

Fig. 4.22 J and volver_J functions

4.7.2. RC Commands

As explained in the chapter of the individual drone, algorithms were created to perform movements that are not implemented in the SDK or that are limited.

Due to the little robustness that the drones have shown when receiving and executing commands, and because these RC commands need the 'stop' command to stop applying. It was ruled out to use them in swarm mode to avoid accidents.

CONCLUSIONS

All tests have been performed at the 'Universitat Politècnica de Catalunya' in two different laboratories. The investigation process and the detection tests for which it was necessary to use a more powerful computer have been developed in laboratory 21 where the computer with the 'GeForce GTX 1060 6GB' graphics card is. The tests of the different applications and scripts have been carried out in the laboratory adapted for the use of drones.

First, an investigation was carried out of which were the possibilities offered by the drone Tello EDU. It was observed that its greatest benefit was in the field of education, which is why it was decided with the academic tutors to focus the project towards this field, to have some bases for future developers to start programming. In addition, we decide to choose a more complete application to make the project more extensive. This application consists of controlling a swarm of drones through body movements. During the research, a GitHub [1] was found, that GitHub contained a series of scripts that allowed the control of a single drone through body movements. These scripts were the basis for my learning of how to program the drone.

In the beginning, my goal was to use Windows as an operating system, however the installation of the different libraries necessary for the use of the found scripts was not optimal, and I decided to use Linux, since the installation of libraries is easier. To install Linux I had to partition my laptop; I had to repeat this process 2 more times because of some error that I still do not know, after a while, I was not allowed to access to the Linux partition. The first time this happened to me, I lost a lot of what I had advanced, from this failure I kept saving my progress every day.

My first objective with these scripts was to control the individual drone with body movements, at first it seemed a simple task since the scripts were supposed to be functional, however, the use of my laptop with the Open Pose library [3] gave very bad results in terms of response time by the drone. During the following days, I was conducting different tests to see how this could be improved. To do this I used different keypoint body models and I was testing with different InWidth and InHeight values. The results were still bad, so we thought about the possibility of using another more powerful computer; the results were better but not as good as expected. That computer was being used by another partner daily, so taking it to the laboratory adapted for drone use, was not an option. It was then decided to change the method of capture body movements. This new method consisted of the detection of colors through the OpenCV library [4], using 3 different color cards, placed one in the right hand, another in the center of the chest and another in the left hand. The biggest problem in this case is that during the day, the light varies and at some time during the day, the colors were not recognized.

Since initially I only had one drone, until I had all four and I could adapt it, I dedicated myself to making different applications for the control of the individual drone. The most productive appears in the document.

Having the four drones, the first thing I did was to understand the scripts already developed in a GitHub [2] to send orders to the drones and adapt it to the graphical interface that I had used for the individual drone, developing different applications for its control.

At the time of adapting the main application for the swarm of drones, a decisive fault (fault of the drone developers) was found. When drones are set into 'AP mode' they cannot receive video. That is why the solution that was thought is to use a drone only to capture the video and the other 3 drones would perform the movements. A Wi-Fi adapter was used to connect the computer to two different networks (the router and the individual drone). At this time, routing problems were found; only one of the two connections worked despite being connected to the two networks. After research through different forums, this problem was solved.

It is expected that in the future, the developers of the drone, will allow to receive video when they are in 'AP mode' what would have been very useful for this project and would allow future developers to make applications that are currently impossible, as it could be the recognition of elements in a warehouse with a swarm of drones.

Another line that can be improved in the future is returning to the capture of movements with the OpenPose library. If you have a sufficiently powerful computer you can get many more positions than those achieved by the method of color detection, since the latter only has 3 marks to analyze, while with OpenPose can go from 15 to 25 marks, being possible to create different positions also for the legs for example.

With the basic missions developed in this document and the explanation of the different commands that can be sent to the drone (all these commands are found in the Tello EDU User Guide[11]), they allow anyone who does not have prior knowledge with Tello EDU to be able to start from there to program more elaborate applications.

The impact of this document is to allow future developers to develop applications in a simple way without having to do extensive research on the internet, since being so new this drone there is little documentation to meet this challenge and it is much diversified in different sites.

ACRONYMS

LED	Light Emitting Diode
EIRP	Effective Isotropic Radiated Power
USB	Universal Serial Bus
LiPo	Lithium-ion Polymer
HD	High Definition
FPS	Frames Per Second
MP	MegaPixels
SDK	Software Developer Kit

REFERENCES

- [1] (2019) Tello-Python Github. [online] Available: <https://github.com/dji-sdk/Tello-Python>
- [2] (2019) Multi-Tello-Formation Github [online] Available: <https://github.com/TelloSDK/Multi-Tello-Formation>
- [3] (2018) Open Pose Github. [online] Available: <https://github.com/spmallick/learnopencv/tree/master/OpenPose>
- [4] (2019) OpenCV website. [online] Available: <https://opencv.org/>
- [5] (2019) Parrot website. [online] Available: <https://www.parrot.com>
- [6] (2019) Robolink website. [online] Available: <https://www.robolink.com>
- [7] (2019) Flybrix website. [online] Available: <https://flybrix.com>
- [8] (2019) Drobots website. [online] Available: <https://drobotscompany.com/>
- [9] (2019) Campuse website. [online] Available: <https://campuse.ro/>
- [10] (2019) Tellopilots Forum. [online] Available: <https://tellopilots.com/>
- [11] (2019) SDK 2.0 User Guide. [online] Available: <https://dl-cdn.ryzerobotics.com>
- [12] (2019) Ryze website [online] Available: <https://www.ryzerobotics.com/es/tello-edu/specs>
- [13] (2019) War on the rocks website [online] Available: <https://warontherocks.com/2019/02/drones-of-mass-destruction-drone-swarms-and-the-future-of-nuclear-chemical-and-biological-weapons/>

Annex A: GitHubs

<https://github.com/hanyazou/TelloPy> -> tracking, image effects

<https://github.com/DevconX/Tello-Python> -> face recognition

<https://github.com/Ubotica/telloCV> -> identify a ball in the scene

https://github.com/markwinap/TensorFlow-Tello-Object_Detection -> detect objects in real time

<https://github.com/piconewton/mindwave-mqtt-tello> -> control with neurowski

https://github.com/D-M-Moriarty/EEG_FYP -> control with neurowski

<https://github.com/NatholBMX/Python-Tello-Control> -> tracking face and hands

<https://github.com/find1dream/Tello-Aruco> -> aruco to estimate the position

<https://github.com/kodamap/tellooo> -> tracking color, face analytics

<https://github.com/HiromuKato/TelloFlute> -> control with a flute

<https://github.com/DroneDance/tello> -> dances

<https://github.com/edwinwongy/TelloSelfie> -> recognize gestures and take photos

<https://github.com/MuAuan/Tello> -> video tracking

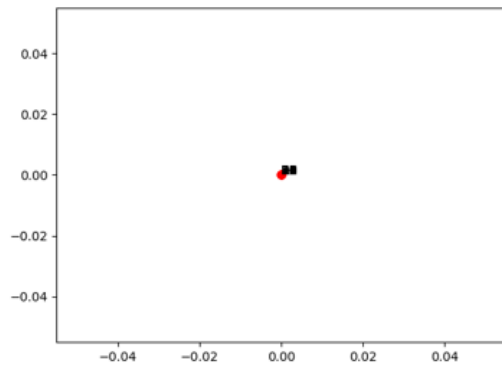
<https://github.com/scepter914/TelloFollowingPerson> -> follow person

<https://github.com/Wason-Fok/Tello-Drone#tello-drone-2019111> -> kinetic

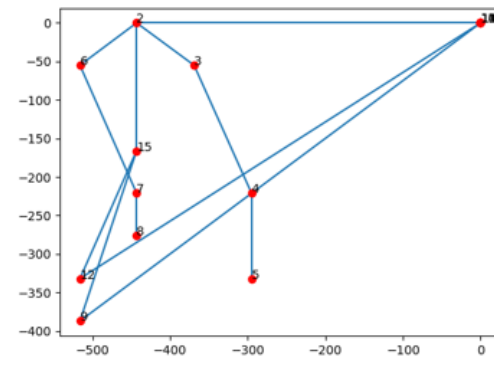
https://github.com/TomoyaFujita2016/Tello_SLAM/blob/master/depth_estimation/depth_est_tello.py -> Depth Prediction with Fully Convolutional Residual Networks

https://github.com/morabrandoi/follow_drone -> follow person, ia, video analysis

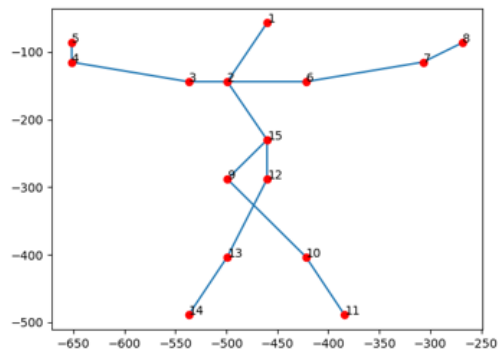
Annex B: Accuracy sample figures



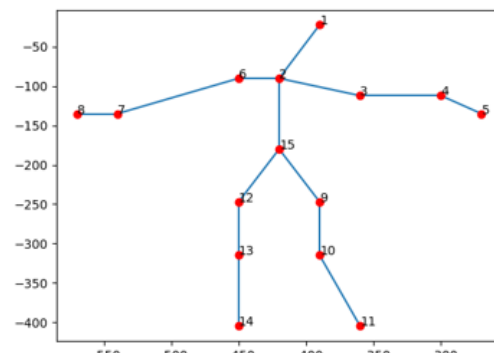
Null



Bad



Acceptable



Very good

Annex C: Org_Coord function

```
def org_coord(self, Lista_drones, Lista_pnts_fin):

    comandos_horiz = []
    drones_horiz = []
    comandos_vert = []
    drones_vert = []
    comandos_esp = []
    comandos_vuelta = []
    drones_esp = []
    Lista_pnts_ini = []

    if len(Lista_drones) is 4:
        P01 = self.Lista_pnts[0]
        P02 = self.Lista_pnts[1]
        P03 = self.Lista_pnts[2]
        P04 = self.Lista_pnts[3]

        Pf1 = Lista_pnts_fin[0]
        Pf2 = Lista_pnts_fin[1]
        Pf3 = Lista_pnts_fin[2]
        Pf4 = Lista_pnts_fin[3]
    elif len(Lista_drones) is 3:
        P01 = self.Lista_pnts[0]
        P02 = self.Lista_pnts[1]
        P03 = self.Lista_pnts[2]

        Pf1 = Lista_pnts_fin[0]
        Pf2 = Lista_pnts_fin[1]
        Pf3 = Lista_pnts_fin[2]

    Lista_pnts_ini = self.Lista_pnyd

    Lista_pnts_ini.pop(0)
    P01x = P01[0]
    P02x = P02[0]
```

```
for n in Lista_pnts_ini:

    var_v = 0
    if P01x = n:

        obstaculo_v[var_v] = n
        var_v = var_v + 1

    else:

        Lista_pnts_ini.pop(0)

        for n in Lista_pnts_ini:

            if P02x = n:

                obstaculo_v[var_v] = n
                var_v = var_v + 1

Lista_pnts_fin.pop(0)
Pflx = Pf1[0]
Pf2x = Pf2[0]

for n in Lista_pnts_fin:

    var_h = 0
    if Pflx = n:

        obstaculo_h[var_h] = n
        var_h = var_h + 1

    else:

        Lista_pnts_fin.pop(0)

        for n in Lista_pnts_fin:

            if Pf2x = n:

                obstaculo_h[var_h] = n
                var_h = var_h + 1
```

```
if (2 in obstaculo_v) or (2 in obstaculo_h):

    drones_espera.append(2)
    comandos_esp.append('forward 100')
    comandos_vuelta.append('back 100')

if (3 in obstaculo_v) or (3 in obstaculo_h):

    drones_espera.append(3)
    comandos_esp.append('forward 100')
    comandos_vuelta.append('back 100')


if 1 in Lista_drones:
    comando_horiz1, comando_vert1 = self.move_to_Point(P01,Pf1)
    self.Lista_pnts[0] = Pf1
else:
    comando_horiz1 = 'None'
    comando_vert1 = 'None'

if 2 in Lista_drones:
    comando_horiz2, comando_vert2 = self.move_to_Point(P02,Pf2)
    self.Lista_pnts[1] = Pf2
else:
    comando_horiz2 = 'None'
    comando_vert2 = 'None'

if 3 in Lista_drones:
    comando_horiz3, comando_vert3 = self.move_to_Point(P03,Pf3)
    self.Lista_pnts[2] = Pf3
else:
    comando_horiz3 = 'None'
    comando_vert3 = 'None'

if 4 in Lista_drones:
    comando_horiz4, comando_vert4 = self.move_to_Point(P04,Pf4)
    self.Lista_pnts[3] = Pf4
else:
    comando_horiz4 = 'None'
    comando_vert4 = 'None'
```

```
if comando_vert1 is not 'None':
    drones_vert.append(1)
    comandos_vert.append(comando_vert1)

if comando_vert2 is not 'None':
    drones_vert.append(2)
    comandos_vert.append(comando_vert2)

if comando_vert3 is not 'None':
    drones_vert.append(3)
    comandos_vert.append(comando_vert3)

if comando_vert4 is not 'None':
    drones_vert.append(4)
    comandos_vert.append(comando_vert4)

if comando_horiz1 is not 'None':
    drones_horiz.append(1)
    comandos_horiz.append(comando_horiz1)

if comando_horiz2 is not 'None':
    drones_horiz.append(2)
    comandos_horiz.append(comando_horiz2)

if comando_horiz3 is not 'None':
    drones_horiz.append(3)
    comandos_horiz.append(comando_horiz3)

if comando_horiz4 is not 'None':
    drones_horiz.append(4)
    comandos_horiz.append(comando_horiz4)

if (len(obstaculo_h) is not 0) or (len(obstaculo_v) is not 0):
    self.save_action2(drones_esp, comandos_esp)

    time.sleep(1)

self.save_action2(drones_vert, comandos_vert)

time.sleep(1)
self.save_action2(drones_horiz, comandos_horiz)

if (len(obstaculo_h) is not 0) or (len(obstaculo_v) is not 0):
    self.save_action2(drones_esp, comandos_vuelta)
```